Konstantinos Larkou 1011182

EPL341 Homework 1

Implementation:

This code appears to implement the A* search algorithm in Java to find the shortest path between two points in a 2D grid. The main method reads a grid and starting and ending positions from a file, then calls the search method to find the shortest path.

The search method uses a PriorityQueue to explore the grid in order of estimated cost, and uses the Manhattan distance heuristic to estimate the distance from each explored node to the end node. The getNeighbors method returns a list of valid neighbor nodes for a given node, where a neighbor is considered valid if it is inside the grid and is not a bomb.

The reconstructPath method takes in the end node and follows the parent nodes back to the start node to reconstruct the shortest path, which is returned as an ArrayList of nodes. The code then writes the length of the shortest path to a file.

Heuristics:

The Manhattan distance heuristic is a measure of the distance between two points in a grid-like path, computed as the sum of the absolute differences of their x and y coordinates. In the context of A* search, the Manhattan distance heuristic estimates the distance between the current node and the goal node by counting the number of moves required to reach the goal node while moving only vertically and horizontally.

```java
// The Manhattan distance is defined as the sum of the absolute differences of
// their row and column coordinates.
private static int manhattanDistance(int row1, int col1, int row2, int col2) {
    return Math.abs(row1 - row2) + Math.abs(col1 - col2);
}
```

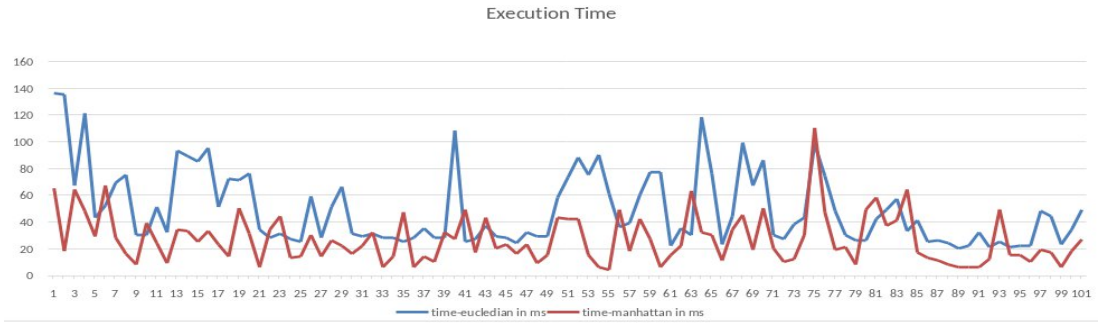
The Euclidean distance heuristic is a measure of the straight-line distance between two points in Euclidean space. In the context of A* search, the Euclidean distance heuristic estimates the distance between the current node and the goal node as the length of the straight line that connects the two nodes. This heuristic assumes that movement is possible in any direction, not just horizontally and vertically, and can provide more accurate distance estimates in some scenarios.

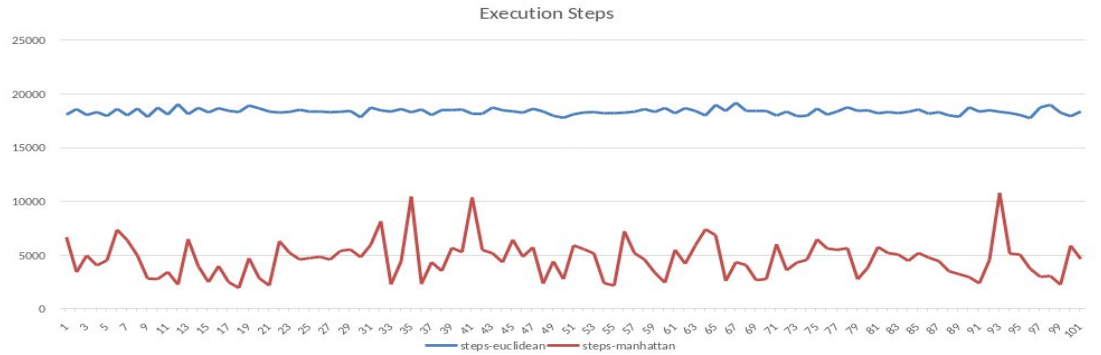
```java
// the Euclidean distance between two points in Euclidean space is the length of
// a line segment between the two points.
private static double euclideanDistance(int row1, int col1, int row2, int col2) {
    return Math.sqrt(Math.pow(row2 - row1, 2) + Math.pow(col2 - col1, 2));
}
```

Konstantinos Larkou 1011182

*** I executed AStar in 100 randomly generated mazes of dimensions 99x99 using two heuristics ***

Graphs:





Euclidean Distance calculates the straight-line distance between two points in a two-dimensional plane, while Manhattan Distance calculates the distance between two points by summing the absolute differences of their coordinates.

Based on the results obtained from running A* using these two heuristics, we can compare their performance in terms of the time taken to find a solution and the number of nodes explored to reach that solution.

From the results, it is evident that the Manhattan Distance heuristic outperforms the Euclidean Distance heuristic in both time and steps taken. The Manhattan Distance heuristic consistently found solutions faster and with fewer steps, as seen from the lower average time and step values across all maze instances.

One reason for this is that the Manhattan Distance heuristic is more accurate in estimating the actual cost of reaching the goal. The Manhattan Distance heuristic, being a less aggressive estimate, is less likely to overestimate the distance to the goal, leading to a more optimal path. In contrast, the Euclidean Distance heuristic can be more aggressive, leading to overestimation and suboptimal paths.

Another possible reason is that the Manhattan Distance heuristic is more suitable for mazes with straight walls and orthogonal paths, which is the case in the given maze. The Euclidean Distance heuristic may be better suited for mazes with curved walls or diagonal paths, where the straight-line distance between points may be a more accurate measure of distance.

Overall, based on the results obtained, it is recommended to use the Manhattan Distance heuristic for solving mazes with straight walls and orthogonal paths, as it provides a more optimal solution in less time and with fewer steps.

Konstantinos Larkou 1011182

Table:

	A	B	C	D	E
1	Maze	time-eucledian in ms	steps-euclidean	time-manhattan in ms	steps-manhattan
2	99x99_0	137	18161	66	674
3	99x99_1	136	18678	19	348
4	99x99_2	68	18121	65	503
5	99x99_3	122	18397	49	411
6	99x99_4	44	18029	30	458
7	99x99_5	53	18697	68	743
8	99x99_6	70	18087	29	646
9	99x99_7	76	18730	17	503
10	99x99_8	31	17936	9	288
11	99x99_9	31	18816	40	284
12	99x99_10	52	18167	25	350
13	99x99_11	33	19138	10	231
14	99x99_12	94	18200	35	655
15	99x99_13	90	18794	34	404
16	99x99_14	86	18364	26	255
17	99x99_15	96	18756	34	405
18	99x99_16	52	18511	24	255
19	99x99_17	73	18397	15	201
20	99x99_18	72	19011	51	478
21	99x99_19	77	18744	32	292
22	99x99_20	35	18442	7	223
23	99x99_21	29	18352	35	637
24	99x99_22	32	18412	45	528
25	99x99_23	28	18615	14	466
26	99x99_24	26	18431	15	480
27	99x99_25	60	18454	31	491
28	99x99_26	29	18379	15	465
29	99x99_27	52	18430	27	546
30	99x99_28	67	18498	23	559
31	99x99_29	32	17906	17	487
32	99x99_30	30	18818	23	603
33	99x99_31	32	18552	33	822
34	99x99_32	29	18440	7	235
35	99x99_33	29	18686	15	450
36	99x99_34	26	18373	48	1052
37	99x99_35	29	18646	7	238
38	99x99_36	36	18116	15	441
39	99x99_37	29	18590	11	359
40	99x99_38	29	18582	33	574
41	99x99_39	109	18635	28	534
42	99x99_40	26	18240	50	1045
43	99x99_41	28	18233	18	555

	A	B	C	D	E
61	99x99_59	78	18776	7	2510
62	99x99_60	23	18287	16	5558
63	99x99_61	36	18767	23	4271
64	99x99_62	31	18504	64	5970
65	99x99_63	119	18073	33	7481
66	99x99_64	77	19071	31	6924
67	99x99_65	24	18500	12	2671
68	99x99_66	45	19271	35	4421
69	99x99_67	100	18516	46	4142
70	99x99_68	68	18507	20	2737
71	99x99_69	87	18503	51	2852
72	99x99_70	31	18067	21	6083
73	99x99_71	28	18438	11	3662
74	99x99_72	39	18023	13	4376
75	99x99_73	44	18039	31	4633
76	99x99_74	99	18728	111	6543
77	99x99_75	75	18151	48	5685
78	99x99_76	49	18460	20	5563
79	99x99_77	31	18846	22	5716
80	99x99_78	27	18511	9	2823
81	99x99_79	27	18562	50	3902
82	99x99_80	43	18282	59	5828
83	99x99_81	50	18399	38	5264
84	99x99_82	58	18298	42	5126
85	99x99_83	34	18417	65	4538
86	99x99_84	42	18644	18	5270
87	99x99_85	26	18242	14	4841
88	99x99_86	27	18391	12	4516
89	99x99_87	25	18082	9	3571
90	99x99_88	21	17963	7	3299
91	99x99_89	23	18846	7	3021
92	99x99_90	33	18435	7	2444
93	99x99_91	22	18567	13	4627
94	99x99_92	26	18421	50	10881
95	99x99_93	22	18305	16	5206
96	99x99_94	23	18129	16	5130
97	99x99_95	23	17830	11	3800
98	99x99_96	49	18833	20	3051
99	99x99_97	45	19066	18	3125
100	99x99_98	24	18338	7	2317
101	99x99_99	35	18008	19	5952
102	Averages	49.86	18442.99	27.71	4710.74

The average time taken by the Manhattan heuristic is significantly lower than the Euclidean heuristic (27.71ms vs. 49.86ms). This suggests that the Manhattan heuristic is able to find the optimal path more efficiently than the Euclidean heuristic.

Similarly, the average number of nodes explored by the Manhattan heuristic is much lower than the Euclidean heuristic (4710.74 vs. 18442.99). This means that the Manhattan heuristic is able to explore fewer nodes in the search space while still finding the optimal path, which is a clear indication of its efficiency.